Performance Analysis of Generative Adversarial Networks

Enrique J. Gonzalez Mendez
University of Puerto Rico at Mayagüez, Mayagüez, P.R., enrique.gonzalez6@upr.edu

Dr. Reena R. Patel
ERDC, ITL, Vicksburg, Mississippi, U.S, reena.r.patel@erdc.dren.mil

July 19, 2019

Abstract
In recent years, generative adversarial networks have shown incredible capacities in the AI and deep learning fields. From generating realistic human faces and bedrooms, to creating realistic photos from only text. We compare various architectures from different types of GANs to see how to use them for simulating possible outcomes of an experiment, which in turn could cut cost and speed up the research and generating missing data and labels. We use various datasets, both image and text based and compare the accuracy of each model and its loss. This is preliminary to later use GANs on a physic-based problem from an ERDC dataset.

Keywords: Deep Learning, Generative Adversarial Networks, AI, Machine Learning, GAN

1. Introduction

Running simulations of real-world problems is computationally intensive and time-consuming task. Setting up the simulation, having the computational resource and the time it takes are daunting and difficult to do often. We propose an approach with generative adversarial networks to run a prediction of how the simulation would go in physic base problems and how they can even help to generate data in fields that do not have enough data. Generative adversarial networks (GANs) have shown incredible results in the field of computer vision and show a lot of potential uses in other fields. A lot of progress has been done in the deep learning area and we can explore these advancements to see how they can help to reduce these daunting tasks.

1.1 Definition

Thanks to researchers in recent years, various types of GANs have been developed to tackle problems from different perspectives. Here we will work with 4 of these models.

1.1.1 GAN

The original GAN as propose in (Goodfellow et al, 2014). The architecture is based on a minmax game between two neural networks, one named the generator and the other the discriminator. The later will output the probability of the input being part of the real data classifying it as real or generated. GANs play a minmax game until arriving to a Nash equilibrium between the two models. Note that all the other models are just variations of this same architecture.

1.1.2 CGAN

The conditional GAN introduces an additional input to the GAN. This additional input will label the data categorically, in most cases this has reported to help the training process and results in better quality images while also classifying each one, giving the user more control over the output (Mirza 2014).

1.2.3 DCGAN

One of the most successful types of GANs. It consists mainly of convolution layers without dense layers or max pooling. It uses convolutional strides and transpose convolutions for down-sampling and up-sampling the data (Radford and Metz 2016). It is mostly used the help identify spatial correlations in the data.

1.2.4 WGAN

The Wasserstein GAN has a peculiar characteristic that differentiates it from the rest of the models. Instead of having a discriminator that gives the probability of the data being real or not, the WGAN has a critic network that will give a score that tries to indicate the "realness or fakeness" of a given image. It also uses a special loss function called Wasserstein loss function (Gulrajani 2017).

2. Approach

We'll use various classes of GANs on different datasets where we can compare the accuracy, resources and time that it takes to effectively train the GANs in respect to the problem. After this, do hyper parameter tuning to see what parameters help most and select the best performing models to do further research on how to improve them and use them on different datasets. There are two types of datasets that will use to do comparisons. Image base datasets and text base datasets (CSV).

3. Process

The process of preparing the models and training can be extensive, here we tried to maintain it as simple as possible to be able to replicate the results in any case or at least have a similar output.

3.1 Pre-processing data

The data for the image base datasets will be all normalize between 0 and 1 or -1 and 1 depending on the architecture that the data will be fed. The data that is text base (CSV) will be divided into two separate sets. One set will be the labels (if needed) and the other will contain the rest of the columns. Checking the size of the data is important to pick out the ideal batch size for training the GANs and number of iterations. For the csv data, the data is worked on TFrecords and h5 data format that will contain the data associated with the steady state flow interaction.

3.2 Architecture

To pick and built the architecture it is very important to investigate the nature of the problem and data. Is the problem a classification problem? Are the data images or something else? We can break down how to build our model picking the right tools for the job.

3.2.1 Layers

The input to the generator will be noise of a certain size, the numbers of perceptrons (neurons) in each layer are chosen in terms of a 2^n but this can be a trial and error process too. The important part is that the generator's output should match the image or the data's dimensions. The discriminator's input will be a layer that expects this dimension and the output layer will be a single perceptron that will tell if the image is real or generated.

3.2.2 Activation Functions

The activation functions play a vital role in the model's architecture. They are the responsible of transforming the output from one layer to another. The activation function that will be our default, thanks to its good reputation, will be the ReLU (rectified linear unit) and its variant Leaky ReLU.

$$ReLu = f(x) = \{0,\ x < 0\ x,\ x \geq 0$$

$$Leaky\ ReLU = f(x) = \{\propto x,\ x < 0\ x,\ x \geq 0$$ where α is typically 0.01

There are two more activation functions that have proven to be very effective in the generator's output and the discriminator's output and for good reasons.

$$sigmoid = f(x) = \frac{1}{e^{-x}}$$

$$\tanh tanh = f(x) = \frac{2}{1 - e^{-2x}} - 1$$

The sigmoid function is assigned to the output of the discriminator to give the probability of the input being classified as generated or real data thanks to it existing between 0 to 1. This function is differentiable so we can find its slope easily.

3.2.3 Loss Functions

The most used loss functions for deep learning are:

$$crossEntropy = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

We will used one more than has been recently develop, the Wasserstein loss. This loss function approximates the earth mover distance.

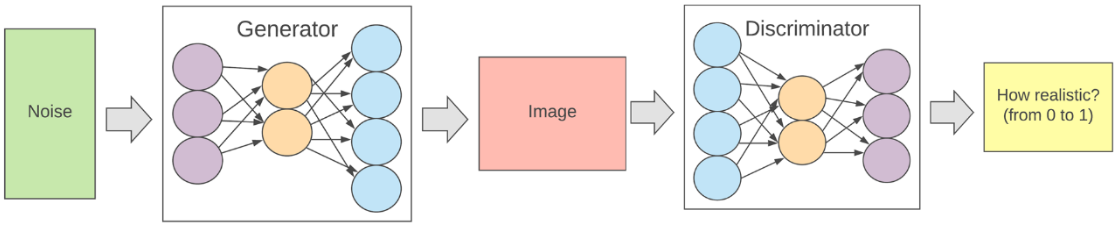
$$\text{Wasserstein} = \underset{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}{\mathbb{E}} [D(\tilde{\boldsymbol{x}})] - \underset{\boldsymbol{x} \sim \mathbb{P}_r}{\mathbb{E}} [D(\boldsymbol{x})] + \lambda \underset{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}} [(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2]$$

3.2.4 Optimizers

The optimizers that we will be using are the Adam optimizer and the RMSPrompt optimizer which are standard for most neural networks.

3.3 Training

Figure 1: General Process of GAN



3.3.1 Algorithm

The algorithm consists of a minmax game between the discriminator (inner loop) tries to maximize the gradient and the generator (outer loop) minimizing the gradient (Goodfellow 2014).

```
for number of training iterations do
    for k steps do
        - sample minibatch of $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$
        - sample minibatch $\{x^{(1)}, ..., x^{(m)}\}$ from data generating distribution $p_{data}(x)$
        - update the discriminator by ascending its stochastic gradient

    end for
```

```
        - sample minibatch of $\{z^{(1)}, ..., z^{(m)}\}$ from $p_g(z)$
        - update the generator by descending its stochastic gradient
    **end for**
```

3.3.2 Details About Training

There is something to note when training the GANs. The real data should be label with a bit of noise (Arjovsky 2017). The discriminator will always be trained first and when the generator is going to be trained (sometimes refer as training the GAN itself) the discriminator's trainability is turn off so that it can be evaluated on classifying the fake and real data. After training the discriminator the labels should be reverse: real data = fake, fake data = real to see the effectiveness of the discriminator's training and how it tackles the images that the generator gives it. After all this, through backpropagation the weights will be updated each epoch. This of course can be manually chosen has the user desires. In various cases discussed (Salimans 2016) the discriminator is train by X steps and the generator by Y steps.

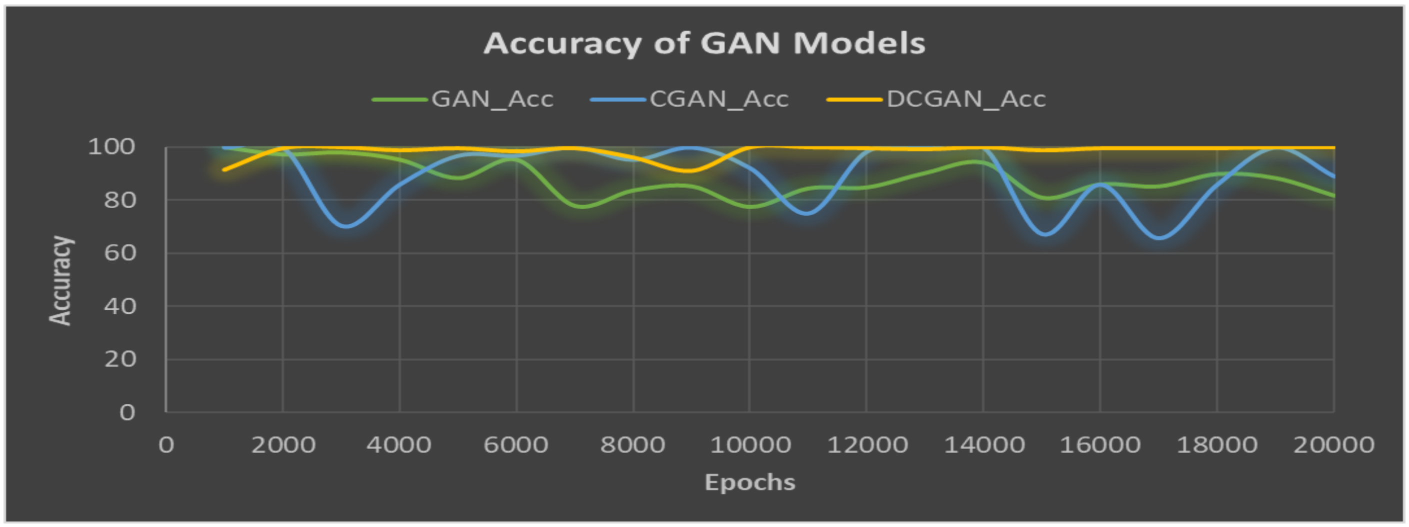
4. Results and Comparisons

4.1 Mnist Data

Here we can observe three of the models that were run with the Mnist dataset. The Mnist dataset contains 60,000 sample of images of handwritten digits. We trained the different models for 20,000 iterations or epochs.

4.1.1 Accuracy

Figure 2: Accuracy of Models



Starting with the "vanilla" GAN (green) we can see that the accuracy is gradually decreasing, which would indicate that the generator is creating better "fake" images of the handwritten numbers. The reason for this is that we want it to reach Nash equilibrium, ideally the discriminator will have a 50% chance to identify of the image is real or generated. The CGAN (blue) seems to be in the same boat, we can infer that those downward spikes are signs that the generator is learning how to generate images like the real ones. The DCGAN (yellow) seems to be identifying correctly almost every time so it may be a mystery to know how the generator is doing against the discriminator, if it is learning or not.

4.1.2 Loss

To understand the training procedure further, we observe the different model's loss function results. The discriminator's loss function will give us an idea of when the GAN itself has learn an important feature in the data and successfully passed the generated image as real data in the view of the discriminator. Following the CGAN we interpret that it learns important features in 3000th, 11000th, 15000th and 17000th iteration. All these plots are good to observe patterns but are not always consistently the same in different runs with the same data and parameters. To counter this, it is good practice to output an image of the data every few iterations. An important note is that the DCGAN's discriminator does not tend to have spikes in comparison to the other models, we could induct that this is one of the reasons that it had a high accuracy as mention previously.

Figure 3: Loss Function of the Discriminator of the Models

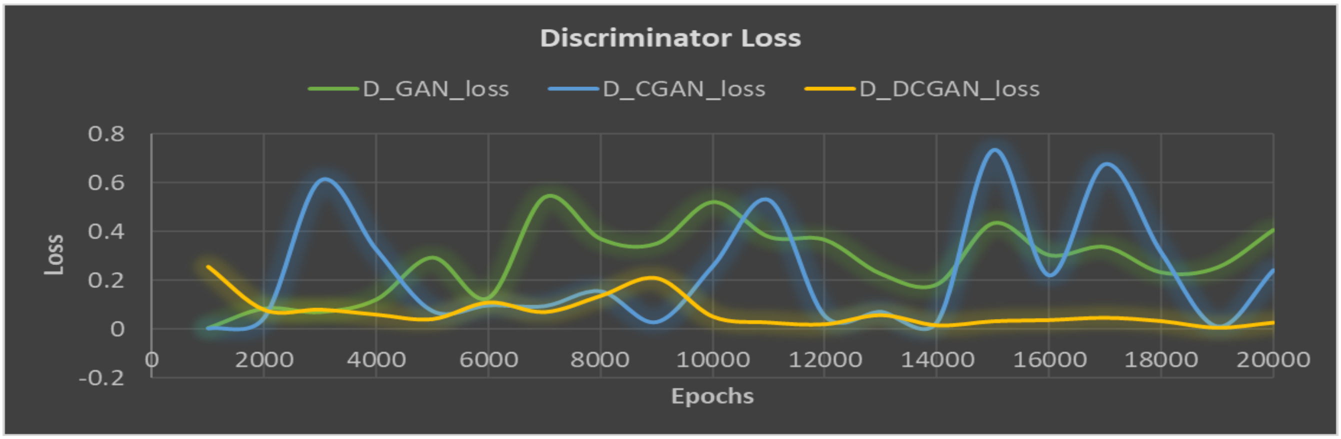
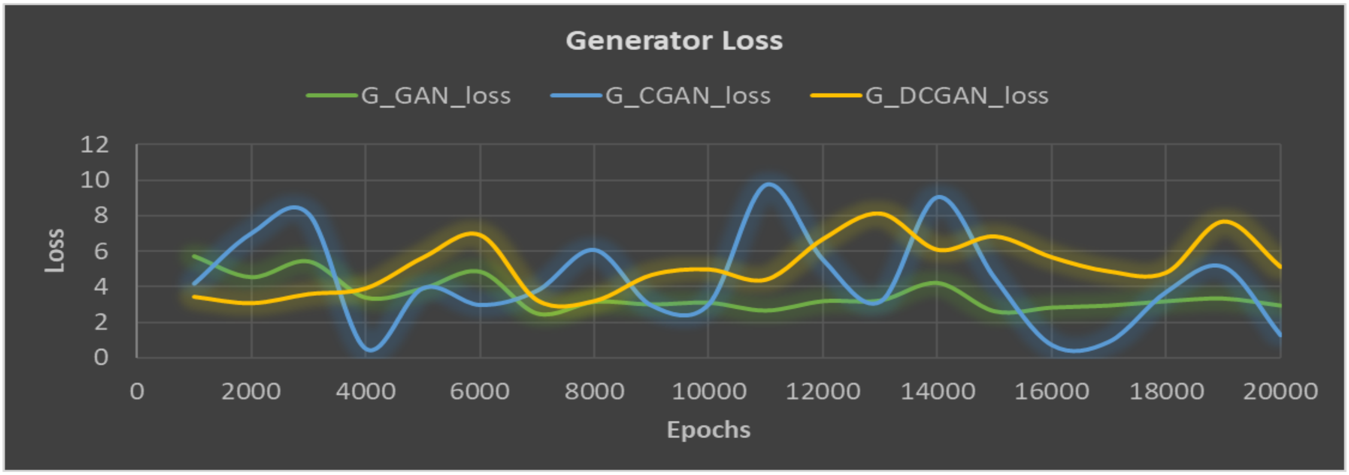


Figure 4: Loss Function of the Generator of the Models



4.1.3 Images

To confirm that the training went well we plot some samples of each GAN. All the images were generated by the GANs. Only the first 1000th iteration and the last iteration will be shown. The input is a random gaussian distribution for all models. Only the CGAN is given an additional output for the classifying opportunity that it brings.

Figure 5: Vanilla GAN Output

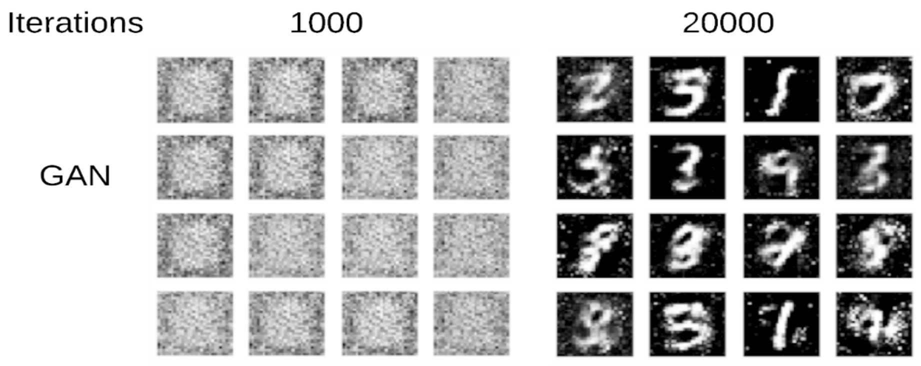


Figure 6: DCGAN Output

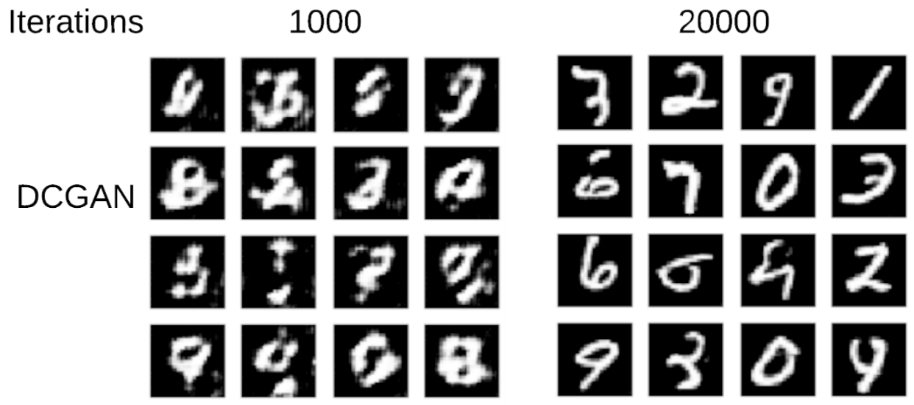


Figure 7: WGAN Output

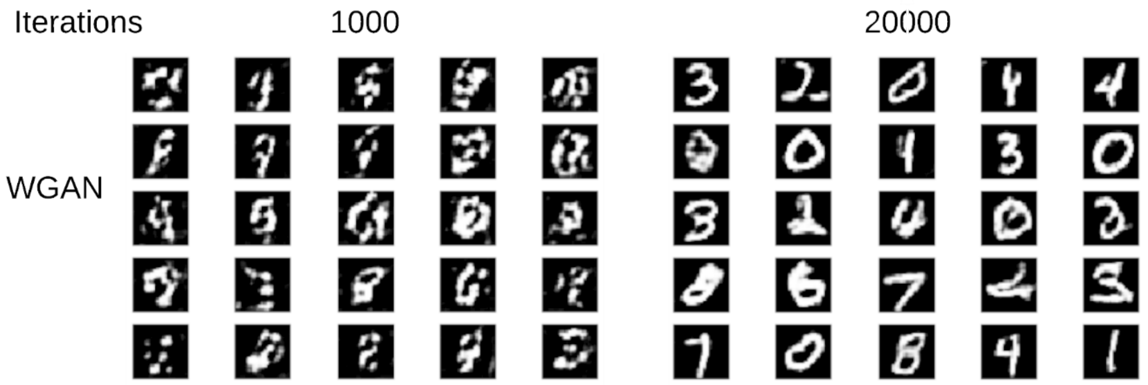
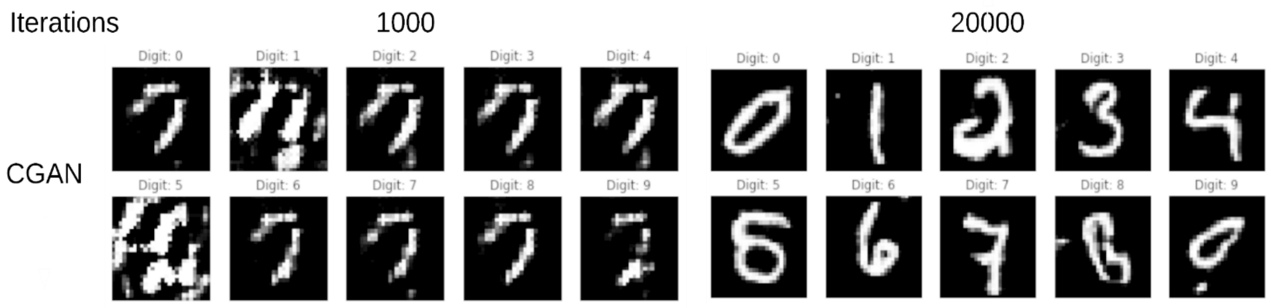


Figure 8: CGAN Output

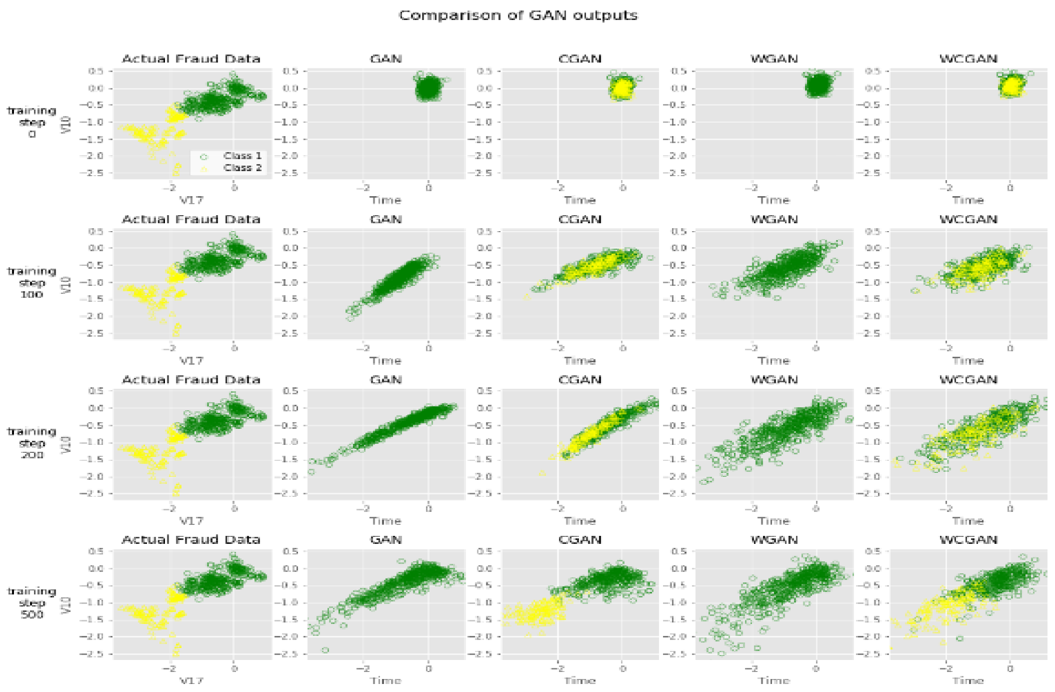


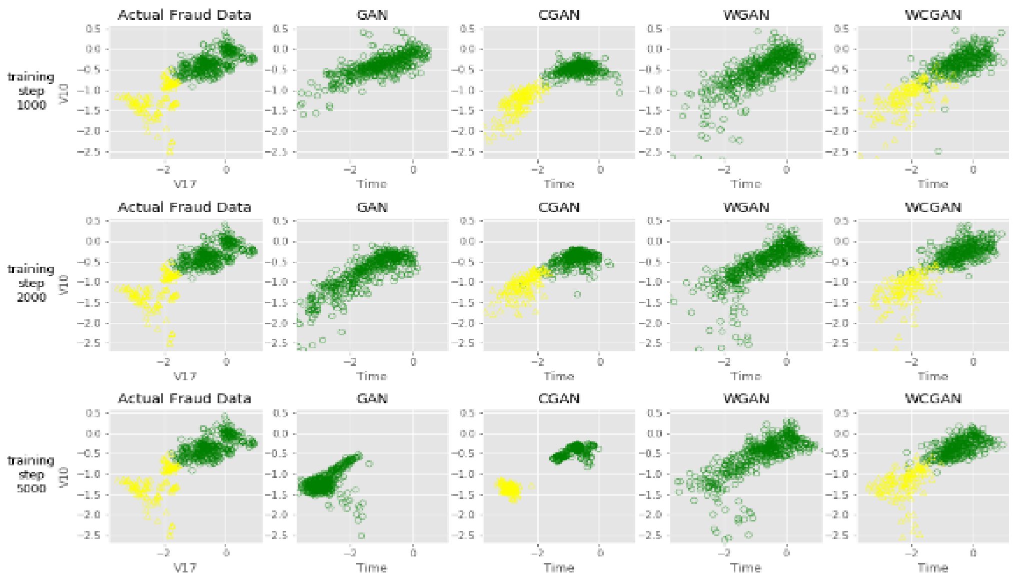
After seeing the output, we can say we have some interesting results. Despite DCGAN's discriminator always having a high accuracy, the generator still learned how to produce the images and seem to be high quality none to less. This shows that the accuracy and loss functions do not always tell the whole story when it comes to GANs and that the output should be checked manually to ensure the best results.

4.2 Credit Card Fraud Data

Here we used the GAN, CGAN and the WGAN and a variation of WGAN to help generate fraud data with only having 492 samples of frauds while the rest are not. This dataset was tested to see if the GANs could be used to generate data in fields where there is not that much data or when the data distribution is very unbalance like in this case: +28000 not fraud, 492 frauds. At first all the models seem to find some relationships in the data and at the 500th step they seem to have similar results. Later, they show that the WGAN is very good at this task in comparison to the GAN and CGAN. After the 2000th step the GAN and CGAN seem to start overfitting the data. This should be due to the Wasserstein loss function and how it manages to give a score to the generator.

Figure 9: Comparison of GAN Outputs in Credit Card Fraud Data





4.3 CFD Data

We use a steady state flow simulation using MECHSYS API following the example of Oliver Hennig (Hennig, 2018) for the convolutional network solution and Jiaz Huoz (Huoz, 2019) for the adversarial network solution. It uses simple geometric figures, in this case various types of circles, squares and triangles, as training samples. After training the networks for a day, they are put on test with different car figures and see how the networks predict the steady state flow interaction with them in comparison with the MECHSYS simulation. The setup is extensive in terms of preparing the images and the steady state flow for the feeding. This can be mediated by researching techniques for preparing data but here we utilize the same procedures made by the authors of the solutions.

Figure 10: CNN vs GAN Car Figure 11

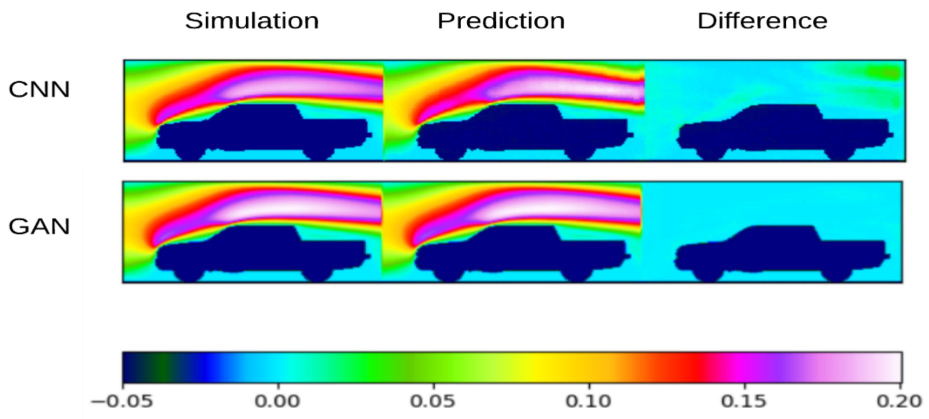
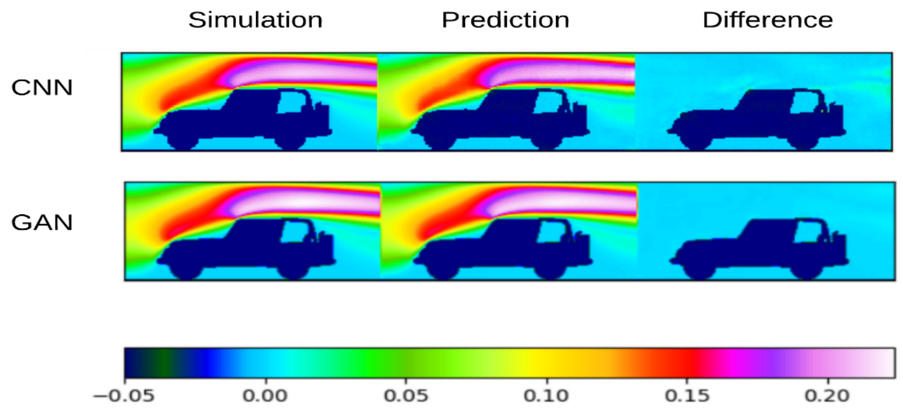


Figure 10: CNN vs GAN Car Figure 8



We see that both networks have a visibly good result. It's interesting to see how the networks has learn the general tendencies of the interactions of a steady state flow and generalized well. The CNN prediction has noticeable errors at the end of the flow while the GAN minimizes the errors. The GAN results are more accurate overall, but the CNN is easier to set up and get running. For sensitive and precise work, the GAN model would be the solution, in case that the margin of error does not need to be so precise, we can use the CNN to have a fair idea of the interaction of the flow. An important next step would be to practice this in a 3D problem and later with a turbulent flow. After analyzing the models, the DCGAN and WGAN variants show promise in physic-based and image-based problems. Further work is needed in the areas of cost functions and general architecture of the GAN. In addition to all, research in the area of data feeding should be done to learn the optimal way for the networks to receive the data and learn quicker the fundamental patterns involved in it.

5. Advantages and Disadvantages

Table 1: Pros and Cons of GAN Models

Models	**Pros**	**Cons**
GAN	The simplicity	Needs to be well design or it will be unstable
CGAN	Better than original GAN but equally as simple	Still is unstable if not set up carefully
DCGAN	High quality results in fewer epochs Identifies spatial correlations quickly	Takes more time to train due to the convolutional layers and multilayers at the end
WGAN	Good training stability Results could be comparable to DCGAN	Difficult to set up

6. Future Work

With this work, and having a solid understanding of how GANs work, the mission now is to give a proof of concept and run a physic base problem related to an ERDC dataset.

7. References

Arjovsky, M. et al, (2017). *Wasserstein GAN*

Goodfellow, I. J. et al, (2014). *Generative Adversarial Nets*

Gulrajani, I. et al, (2017). *Improved Training of Wasserstein GANs*

Hennig, O. et al, (2018). *Steady State Flow with Neural Nets.* GitHub repository: https://github.com/loliverhennigh/Steady-State-Flow-With-Neural-Nets.

Huoz, J. (2019). *GAN Flownet.* GitHub repository: https://github.com/jiazhaozhu/GAN_flownet

Mirza, M. (2014) *Conditional Generative Adversarial Nets*

Radford, A. and Metz, L. (2016). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*

Salimans, T. et al (2016). *Improved Techniques for Training GANs*

Sharma, S. (2017) *Activation Functions in Neural Networks*, 25/06/2019, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6